# 动静结合的网络恶意代码检测技术研究 *

邓兆琨，陆余良，黄　钊

(国防科技大学，合肥 230037)

**摘　要：**针对互联网服务器的攻击行为常利用程序存在的未知漏洞展开，且手法不断更新，这使得防御技术的更新往往长时间滞后于攻击行为的发生。提出了一种动静结合的网络数据检测方法，该方法在传统静态分析的基础上优化了检测算法，同时引入了动态模拟运行的检测方式。通过动静结合的双重检测提高了恶意代码的识别率，并能够在代码传入实际被保护程序运行前检测并确定其恶意性，实现防御系统策略的快速自动化更新，缩短了策略更新时间，提高了防御效果。结合该技术实现了一个 N-A detecting（网络数据检测）防御系统，实验证明，该系统能够很好地防范针对网络程序的恶意代码攻击行为。

**关键词：**恶意代码；数据检测；静态分析；动态运行；网络数据

**中图分类号：**TP311　　doi: 10.3969/j.issn.1001-3695.2018.01.0040

## Research on static and dynamic detection technology of network bad code

Deng Zhaokun, Lu Yuliang, Huang Zhao

(*National University of Defense Technology*, *Hefei* 230037, *China*)

**Abstract:** The attacks against Internet servers usually use unexplored vulnerabilities that exist in the program or the attacks are constantly being updated. All of these make the defensive measures often lag behind the attack. In order to change this problem, this paper presented a static and dynamic network data detection technology, which optimized the detection algorithm on the basic of traditional static analysis, and introduced the dynamic simulation to improve the rate of malicious data detection. It could detect malicious data before it introduced into the protection program and realized automatic update of defense system strategy and optimized the defense effect. This paper combined the technology to implement an N-A (network data detection) defense system. According to the the experiment result, this system can be used to prevent bad code attacks against network programs.

**Key words:** bad code; data detection; static analysis; dynamic operation; network date



## 0　引言

随着互联网技术的不断发展，针对互联网安全问题的研究也变得日益重要，而作为用户使用互联网技术的重要载体，网络程序安全也成为了计算机网络安全的一个重要研究方向[1~4]。由于互联网的开放性，针对网络程序缺陷的攻击行为一直是互联网安全的一个重要隐患，特别是网络程序的服务器端存储的大量敏感数据。恶意攻击者常以网络程序软件为跳板，来达到攻击后台数据库或者下载敏感数据的目的。

网络程序一般指基于网络数据传输或者网络通信提供服务的二进制应用程序。与一般的单端程序软件不同，网络程序由服务器端和客户端两部分组成，通过双端软件间的数据通信、文件传输为客户提供服务。当客户端发出请求数据和服务器端建立连接后，服务器端会通过物理链路以网络流量数据的形式向客户端提供服务。因此，针对网络程序的安全研究也分为了两个方向[5]：一个方向是针对程序自身缺陷的克服，即通过软件补丁的形式对软件本身的已知漏洞进行弥补，从而消除软件两端存在的脆弱点；另一个方向是对到达服务器端的代码数据的检测，即通过在本地设置检测过滤策略的方式，对软件接收到的网络数据进行识别，当判定接收到的数据会导致一些非法操作时则对其进行过滤。这两种思路的综合运用，对网络程序安全提供了巨大支撑。研究证明[6]，两者的结合能够有效克制大部分来自互联网的针对网络程序的攻击。但这两种思路均存在着一定的缺陷。首先，针对自身缺陷的弥补策略往往存在滞后性，研究者对于软件程序本身的缺陷测试集中于开发阶段，而在软件发布后，如果攻击者通过软件分析的方式，挖掘到程序自身存在的未知漏洞并利用该缺陷对程序进行攻击，则安全人员往往很难在短时间内发现该漏洞并加以弥补。例如比较经

**收稿日期：**2018-01-22；**修回日期：**2018-03-09　　**基金项目：**国家重点研发计划重点专项资助项目（2017YFB0802905）

**作者简介：**邓兆琨（1993-），男，安徽合肥人，硕士研究生，主要研究方向为数据流量检测、网络空间安全（dengzhaokun1993@126.com）；陆余良（1964-），男，教授，博导，主要研究方向为计算机网络安全；黄钊（1994-），女，硕士研究生，主要研究方向为信息安全．

典的乌克兰电站攻击事件，就是由于攻击者掌握了电站服务程序的未知漏洞，利用 U 盘摆渡的方式对程序展开攻击，才造成了较大的破坏效果。其次，服务器程序运行过程中的安全防御对安全人员的技术水平依赖较大，针对可疑数据的识别常通过人工静态分析的方式，而无法对数据在程序中动态运行的效果进行自动化判定，这就导致了现阶段某些攻击手段通过过程行为隐藏的方式对自身效果进行隐藏，从而绕过防御系统实现攻击。例如 2017 年发生的针对国内某论坛的攻击行为，经调查，攻击者就是将恶意代码隐藏在了正常的文件中上传到论坛服务器，从而实现攻击目的。

本文主要着眼于网络程序安全中恶意代码检测技术的改进，对该技术现阶段的发展情况进行了研究，通过对该技术自身缺陷的分析，提出了一种动静结合的检测方法，在该方法的基础上形成了一个检测系统，并通过实验验证了系统的可行性、有效性、稳定性。

## 1　相关研究进展

恶意代码检测是网络程序在实际运行过程中的一个重要防护手段[7]，现阶段代码检测的展开主要依托于网络架构中防火墙和用户服务器两个节点实现[8,9]。运行于服务器端的检测系统能够在检测完成后实现合法代码的高效传输，且架构简单，无须额外的复杂硬件支持，但由于其本身和真实服务器程序运行于同一节点，容易被攻击者借助服务器硬件漏洞实现绕过[9]。防火墙作为内外网分割的界限在运行上更具优势[10]，其本身在进行检测工作的同时，能最大程度上减少对服务器节点稳定性的破坏,但作为内网面对外网的直接接口，随着数据流量的增大易造成防火墙负担过重。文献[11]明确了防火墙技术的优势，以及跨平台的优良性能，并基于此提出了在传统防火墙的基础上实施算法更新和最优策略集选择的新思路。

现有的静态检测技术主要通过对恶意代码的特征进行预先设置和存储，利用恶意代码匹配算法对网络数据包中的数据部分进行分析检测，并对确定为恶意代码的数据内容进行过滤[12]。但此类技术对恶意代码的检测常基于固定策略，而对采取新攻击手段或针对未知漏洞的恶意代码的抵抗性往往较差。针对该问题，众多厂商通常会对自身防火墙的防御库进行更新，但该更新通常是在漏洞爆出并对用户造成一定损失后才会进行，存在一定滞后性，且对流量的检测仅基于静态分析和黑名单预置，针对已知漏洞后期出现的新攻击手段防御较差。目前，行业内针对网络程序运行安全已经有了一些比较成熟的防御系统。

针对数据监测在网络程序安全防护过程中的滞后性，许多研究者针对动态检测方式进行了研究。文献[13]则提出了基于行为预测的防御策略，通过对本地高危网络程序和高危函数的监控，预测攻击者可能实施攻击的行为，从而在一定程度上克服了滞后性。文献[14]针对防火墙设计提出了分层比对的方式，即针对静态分析过程针对性不强的缺点，将不同类别或者协议的网络数据分组进行动态检测来提高针对性，同时降低静态分析过程造成的延迟。文献[15]设计了一套下一代防火墙系统，实现了对网络流量的识别与控制，通过采用多种检测技术相结合的方法提高了协议识别的准确率，并依托系统的架构设计出流量控制方案，提出了针对流量控制的限流算法，实现了对具体流量的控制作用。直接在防火墙节点运行网络数据检测的好处是执行开销较小，仅对网络数据部分进行检测[16]，但与其他静态分析技术存在的弊端类似，通过网络数据比对、校验、过滤等流程实现的网络数据静态检测，对分析策略和恶意代码信息库的依赖较大，后期针对新缺陷、新攻击手段的防御需要人工的更新，对研究者的要求较高，且存在滞后性。

基于以上研究，本文首先对静态分析技术进行优化，提高静态检测效率和准确度；之后采用 qemu 虚拟机技术搭建环境实现动态运行检测，该技术通过命令行控制台实现对操作系统运行的控制，仅对目标程序在系统下的运行提供支撑，减少了图形化界面等其他非必须程序运行带来的系统开销，是一种“轻量级”的虚拟机技术。通过两者结合提出了一种静态分析和动态运行相结合的网络数据检测方法，并在该方法的基础上形成了一个基于该技术的网络程序安全防御系统。通过模拟攻击行为的方式，对该系统的防御效果进行了测试。

## 2　动静结合的网络数据监测技术

### 2.1　数据监控和提取

数据监控技术会根据 lua 脚本中的相关配置信息，对接收到的数据进行提取、整理和拼接。数据的提取过程主要分为三步进行：a)在接收到网络中其他主机发来的底层数据后，根据配置信息截取目标数据帧，并交付给以太网驱动；b)本地以太网驱动会根据数据帧首部中的 type 字段（RFC 894）确定该数据帧的协议类型（IP、ARP 或 RARP），并提取有效载荷部分交付给相应的协议；c)按照相应的协议向高层解析，直到将真实数据部分提取出来。图 1 是该过程的示意图。

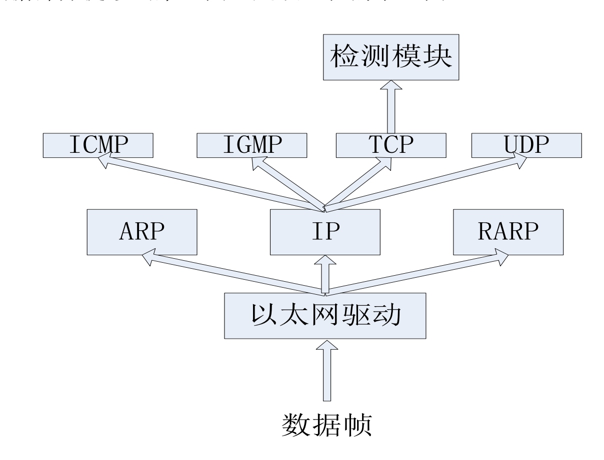



图 1　实际数据提取过程

数据处理模块会自动到指定路径下的数据存储文件中读取流量监控模块提取出的数据信息，对不同数据包所采取的协议类型进行判定，再按照该协议 RFC 文档[17]的指导提取数据包的详细信息（如对实际数据包的 num 和 ID 标志位进行读取），

并对数据通信数据进行重组，最终将底层数据流组合为完整的数据信息。代码 1 为该功能部分的代码。

代码 1　数据提取重组部分

```
while(i<len(string_data)):#循环解析各个数据包包头、数据包数据
        #数据包头各个字段
        pcap_packet_header['GMTtime'] = string_data[i:i+4]
        pcap_packet_header['MicroTime'] = string_data[i+4:i+8]
        pcap_packet_header['caplen'] = string_data[i+8:i+12]#数据包长度
        pcap_packet_header['len'] = string_data[i+12:i+16]#数据包真实长度
        #求出此包的包长len
        packet_len = struct.unpack('I',pcap_packet_header['len'])[0]
        #写入此包数据
        packet_data.append(string_data[i+16:i+16+packet_len])
        i = i+16+ packet_len
        packet_num+=1#数据包个数

#把数据包信息写入result.txt
f = open('/home/dzk/Desktop/1/13.pcap')
pcap = dpkt.pcap.Reader(f)
for (ts,buf) in pcap:
        eth = dpkt.ethernet.Ethernet(buf)
        ip = eth.data
        tcp = ip.data
for i in range(packet_num):
        if(eth.type != 2048):
                continue;
        if tcp.__class__.__name__ == 'TCP':#__class__获得已知对象的类，
                #先写每一包的包头
                ftxt.write("这是第"+str(i)+"包数据的包头和数据："+'\n')
                for key in ['GMTtime','MicroTime','caplen','len']:
                        ftxt.write(key+' : '+repr(pcap_packet_header[key])+'\n')
                #再写数据部分
                ftxt.write('此包的数据内容'+repr(packet_data[i])+'\n')
ftxt.write('一共有'+str(packet_num)+"包数据"+'\n')
```

通过实验证明了该步骤的必要性。因为现阶段针对网络程序的一种重要的攻击隐藏手段就是将恶意代码分割为大小不一的数据段并分别嵌入到数据包中，如果未分割后的同源数据包进行重组和统一匹配，则针对单一数据包内容的静态分析手段将无法识别出拆分后的恶意代码。在实验过程中，笔者将一段溢出代码分割成十段长短不一的数据字符串，并嵌入十个连续数据包中，现有的单一流量过滤策略对其并未实现过滤，从而在一定程度上造成了防御系统的失效。

### 2.2　静态分析检测

静态分析检测主要依托于数据自动化检测技术，对客户端发送到本地服务器端的一段完整数据内容进行检测。该技术检测准确率的提升存在一定瓶颈[18]，这主要是由于检测细致度的提高会导致检测时间的延长，从而引起本地服务器程序接收合法数据的时延变长。不同研究中针对该问题的克服主要通过检测算法的改进来实现，即数据匹配算法时间复杂度的降低。为提高分析效率，本文在改进检测算法的基础上，将静态分析检测过程划分为快速检测和算法比对两个阶段，如图 2 所示，图 2（a）为不经检测的数据接收延迟，（b）为加入粗测技术的两段式检测。可以看出，对于数据接收延迟的影响控制在了“+3 ms/单数据字符串”之内。

通过信息收集的方式，在开发过程中已在系统数据库中预先设置了一定数量的恶意攻击代码，这些恶意代码已经在互联网上出现并造成了破坏。根据不同恶意代码的运行效果建立了三个破坏性效果分类列表，分别是导致系统崩溃数据列表、任意内存读取列表、任意内存写入列表。该环节的输入接口接收得到上一模块处理后的数据，输出接口输出对该数据的判断结果。

第一阶段进行快速比对。该过程基于特征字符定位和字符串匹配，针对攻击 shellcode 数据中常用的填充字符和跳转指令进行匹配查找。例如针对攻击者常在恶意代码中大量填充空字节达到溢出效果的习惯。在快速对比阶段，如果连续重复字节的长度超过指定上限，则将被判定为恶意溢出数据，这样对一部分攻击流量实现粗过滤。

```
来自 [illegible] 的回复: 字节=32 时间=29ms TTL=54
来自 [illegible] 的回复: 字节=32 时间=29ms TTL=54
来自 [illegible] 的回复: 字节=32 时间=30ms TTL=54
来自 [illegible] 的回复: 字节=32 时间=29ms TTL=54
来自 [illegible] 的回复: 字节=32 时间=29ms TTL=54
来自 [illegible] 的回复: 字节=32 时间=30ms TTL=54
来自 [illegible] 的回复: 字节=32 时间=30ms TTL=54
来自 [illegible] 的回复: 字节=32 时间=30ms TTL=54
来自 [illegible] 的回复: 字节=32 时间=29ms TTL=54
```

(a)

```
来自 [illegible] 的回复: 字节=32 时间=32ms TTL=48
来自 [illegible] 的回复: 字节=32 时间=31ms TTL=48
来自 [illegible] 的回复: 字节=32 时间=34ms TTL=48
来自 [illegible] 的回复: 字节=32 时间=33ms TTL=48
来自 [illegible] 的回复: 字节=32 时间=32ms TTL=48
来自 [illegible] 的回复: 字节=32 时间=32ms TTL=48
来自 [illegible] 的回复: 字节=32 时间=32ms TTL=48
来自 [illegible] 的回复: 字节=32 时间=33ms TTL=48
来自 [illegible] 的回复: 字节=32 时间=33ms TTL=48
```

(b)

图 2　网络延迟

第二阶段利用改进的 KMP 算法对经过第一阶段粗过滤的数据进行检测。代码 2 为该算法的部分代码。在这一阶段，根据本地记录的规则截取被检测数据所有的可疑数据段，然后将这些可疑数据段做笛卡尔积构造出超矩形，根据不同超矩形的类别，再同规则本身进行比较，这样，就将一个长数据字符串比较的过程分解为了同时进行的几部分，从而提高了在检测时间上的效率。由于算法在一开始做笛卡尔积运算时会占用较大的额外存储空间，所以随着检测规则库的增大，这种算法所带来的空间开销也会增大。针对该问题作者在目前研究中还未发现较好的解决办法，这也是下一步的研究方向。

代码 2　改进的 KMP 算法比对代码

```
……
void match(string s, string t, int a[])
{
    n=s[0];
m=t[0];
ff(t, f);//求出被检测代码中各个字符的前
//缀函数 ff(x)
d=0;//该变量用来记录匹配成功的字符串长
//度
q=0
for(i=1;i<=n;i++)
{
while(q>0&&t[q+1]!=s[i])
q=f[q];
if(t[q+1]==s[i])
q=q+1;
if(q==m)
{d=d+1; a[d]=i-m+1;}
}
```

```
}
……
```

根据匹配结果的不同分为两种处理方式：

a)当发现被测数据与数据库中某一攻击数据相匹配，则判断为恶意代码，系统向管理员提交检测报告。

b)当未发现匹配或匹配相似度较小，则将该数据自动提交给动态运行检测模块进行执行检测。

### 2.3　动态分析检测

本文在动态分析检测过程中采用了“类沙箱技术”，通过搭建类似被保护服务器的模拟环境，将可疑数据导入该环境下运行，并监控处理效果，从而实现了恶意代码的无危害运行。

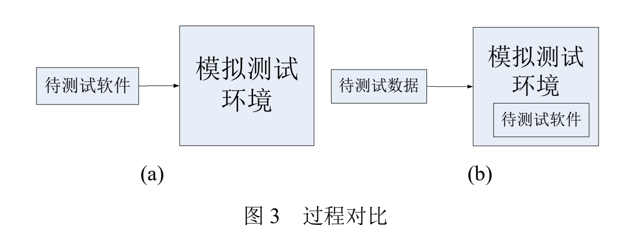



图 3　过程对比

如图 3（a）所示，传统沙箱技术的思想是将待测试软件传入虚拟环境下运行，这样所有操作都是虚拟的，在程序运行过程中，真实的文件和注册表不会被改动，从而确保可疑程序无法对系统关键部位进行改动而破坏系统；如图 3（b）所示，本文采用的动态运行分析过程是建立在已搭建好的 qemu 虚拟机环境下，本地服务器程序的副本已经运行在该虚拟环境下，每次接收到网络上发送来的数据后，系统只是将数据传入该虚拟化环境中，输入给被保护程序的副本并执行。

```
(e64.6d4): Integer divide-by-zero - code c0000094 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000004 ebx=7ffdf000 ecx=00000000 edx=00000000 esi=0012fe2c edi=0012fe08
eip=0040139c esp=0012fcf0 ebp=0012fe20 iopl=0         nv up ei pl nz ac pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010216
*** WARNING: Unable to verify checksum for SEH_test.exe
SEH_test!test+0x7c:
0040139c f7bd1cffffff    idiv    eax,dword ptr [ebp-0E4h] ss:0023:0012fd3c=00000000
0:000> g
(e64.6d4): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=00000000 ecx=31684130 edx=7c9037d8 esi=00000000 edi=00000000
eip=31684130 esp=0012f928 ebp=0012f948 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
31684130 ??              ???

0:000>
Ln 0, Col 0 | Sys 0:<Local> | Proc 000:e64 | Thrd 000:6d4 | ASM | OVR | CAPS | NUM
```

图 4　内存地址监视

如图 4 所示，在执行的过程中挂载调试器，监视 EIP 等寄存器内容的变化，通过代码 3 所示模块实现对服务器程序缓冲区操作函数 API 的挂钩，截取相关调用信号，监控程序执行过程中存在地址跳转、内存读写的关键函数。当服务器程序在处理接收数据的过程中违规跳转到敏感地址，或者对内核敏感区域进行读写操作时，则对跳转地址的内容或者计划写入的内容进行记录，并保存对应时刻状态。

代码 3　重要函数挂钩

```
……
void H_windows_net_api::build_NetAPI_hooks
(FunctionMonitor  * funcMonitor,
S2EExecutionState * state,
uint64_t  targetCR3)
{
FunctionMonitor::CallSignal * callSignal;
m_targetCR3 = targetCR3;

// socket
callSignal = funcMonitor->getCallSignal( state,
socket_vaddr,
targetCR3);
callSignal->connect( sigc::mem_fun( *this,
&H_windows_net_api::socket_call_hook)
);
// bind
callSignal = funcMonitor->getCallSignal( state,
bind_vaddr,
targetCR3);
callSignal->connect( sigc::mem_fun( *this,
&H_windows_net_api::bind_call_hook)
);
// accept
callSignal = funcMonitor->getCallSignal( state,
accept_vaddr,
targetCR3);
callSignal->connect( sigc::mem_fun( *this,
&H_windows_net_api::accept_call_hook
);
// recv
callSignal = funcMonitor->getCallSignal( state,
recv_vaddr,
targetCR3);
callSignal->connect( sigc::mem_fun( *this,
&H_windows_net_api::recv_call_hook)
);
}
……
```

该过程主要实现两项功能，首先是判定数据的合法性，其次是根据判定结果决定对数据的处理方式。如果目标程序对该数据的处理过程会对系统造成损害，或者直接导致程序崩溃，则将该数据标记为恶意代码，并提取其中的主要部分为特征字段，根据其破坏效果的不同加入到脏数据对比模块的支撑数据库中，更新对比策略；如果程序正常运行，并得到了正常的输出结果，则将该数据从系统输出接口输出，并发送到本地环境下正常提供服务的服务器上。通过实验证明，该动态技术能够提高恶意代码检测的成功率，对一些利用未知漏洞或者采取新攻击手段的恶意代码能够实现成功过滤。

## 3 系统架构

本文提出的基于网络数据检测的网络程序防御系统（network-data detecting defense）主要分为四个基本模块，即流量监控模块（flow monitoring module）、数据处理模块（data processing module）、脏数据比对模块（dirty data comparison module）和动态自动化检测模块（dynamic automatic detection module）。图 5 所示为该系统原型架构。

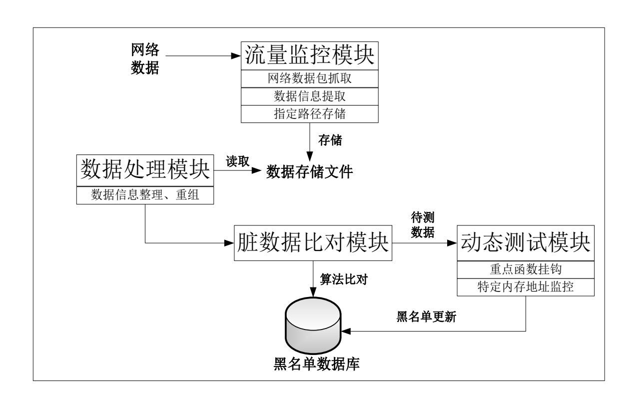



图 5 系统原型架构

该系统的运行建立在互联网基本架构的基础上，网络中客户端主机发送的数据通过底层物理链路以二进制数据流的形式传输到达本地，链路层协议软件将其还原为高层协议数据包并提交给本地的网络程序防御系统。本地防御系统的网络接口端接收到的是底层提交的数据包，而输出接口端输出的是经过完整检测流程的合法数据内容。在正常的数据包逐层解析的基础上，该系统实现了对有效数据载荷的提取和检测功能.系统的数据检测过程主要分为 以下三步：

a)流量数据提取。系统通过网络监听的方式抓取数据流量，该过程基于 wireshark 软件的早期开源代码实现。表 1 所示为系统在本地仿真实验环境下的一次预设信息。通过流量监控模块和数据处理模块的配合，对通信数据中符合预设要求的网络数据包实现监听和提取，按照数据类别和到达顺序将处理后的数据信息保存到指定路径的 txt 文件中。

表 1 流量监控模块配置信息

| 项目名 | 实际参数 |
|---|---|
| The ID of analyzing files | 12 |
| The source IP | 192.168.122.3 |
| The source port | 80 |
| The Destination Port | 80 |

b)数据静态检测。数据对比环节主要由脏数据（bad data）对比模块和相应的支持数据库组成。根据系统内嵌的比对算法，将提取到的流量数据和本地黑名单数据库中的数据进行比对，实现攻击流量的初步筛选和验证。

c)动态运行测试。针对无法确定的流量进行运行验证，通过动态运行测试的方式检验流量数据的实际运行效果，对数据的合法性实现判定。

## 4 实验与分析

为达到模拟实际环境的效果，本文搭建了如图 6 所示的仿真实验环境。数据到达本地防火墙后，可通过路径 I 或 II 进入内网。

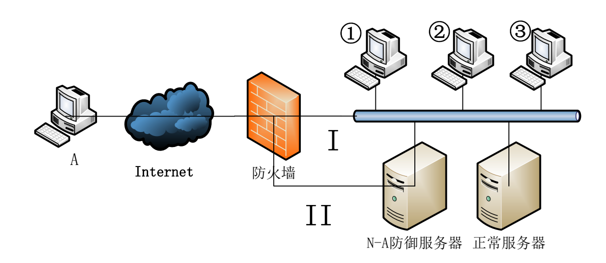



图 6 实验环境架构

该实验环境由内、外网两部分构成，通过防火墙实现隔离。表 2 所示为各个主机与服务器节点相关属性。其中外网主机 A 安装 Ubuntu 14.04 操作系统，拥有独立公网 IP 地址，通过互联网连接到本地网域；本地内网由两台安装 Ubuntu Server 操作系统的服务器和三台用户主机组成，该内网对外共用一个独立公网 IP 地址。

表 2 实验环境配置信息

| 区域 | 名称 | 节点属性 |
|---|---|---|
| 外网 | 主机 A | Ubuntu 14.04 |
| 防火墙 | | McAfee 14.0.9042 |
| | 主机① | Windows 7 |
| | 主机② | Windows 7 |
| 内网 | 主机③ | Windows 7 |
| | N-A 防御服务器 | Ubuntu Server |
| | 正常服务器 | Ubuntu Server |

内网服务器搭载了两款存在典型漏洞的服务器软件，软件版本及漏洞类型如表 3 所示。外网主机 A 模拟恶意攻击者，对内网提供正常服务的服务器发送恶意代码，按照破坏效果划分，恶意代码能造成两款服务器端软件的崩溃，或者能实现对服务器指定内存的非法读写操作。

表 3 目标测试程序及漏洞信息

| 软件名称 | 版本号 | 存在漏洞编号 | 漏洞类型 | 系统类型 |
|---|---|---|---|---|
| Tftp32 | 3.51.0.0 | CVE-2013-6809 | 格式化字符串漏洞 | Windows Server |
| HttpDX | 1.0.1e | OSVDB-ID-84454 | 堆溢出 | Windows Server |

针对两种漏洞分别准备了 300 种恶意代码（100 种溢出、100 种内存读取、100 种内存覆盖）。外网主机 A 作为恶意攻击者，通过正常数据通信的方式向目标服务器发起请求，并将恶意代码夹杂在正常服务请求的中间发送到目标服务器。实验采

用连续发送的方式，将 600 种恶意代码分割后，隐藏在正常的数据通信中发送到目标服务器。

a)仅采取防火墙策略

数据发送到目标网络防火墙之后，通过线路 I 发送到目标服务器。破坏效果实现如图 7 所示。可以看出，单纯防火墙策略并未实现恶意代码过滤，从而导致了目标服务器软件的崩溃；通过内存地址可以看到，对于恶意代码同样能够写入内存相应地址。

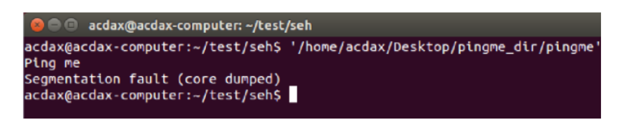



图 7　破坏效果实现

图 8 所示为防火墙策略下，针对两种网络程序的恶意代码攻击防御效果。可以看出，单纯防火墙策略的防御效果较弱，特别是在恶意代码的特征码未更新到防火墙策略库时，针对两款软件漏洞的防御率仅能达到 9.67%和 13.34%。

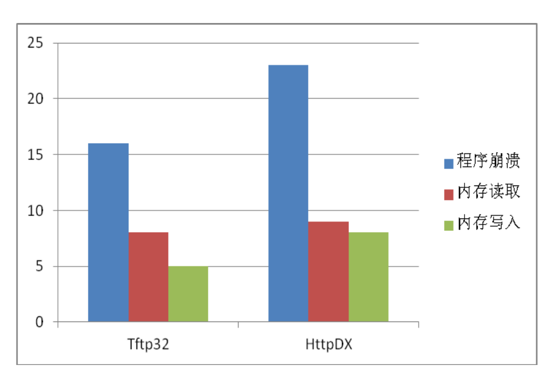



图 8　实验防御效果统计

b)采取“防火墙＋N-A 防御服务器”的策略

根据破坏效果的不同，防御系统会将验证到的恶意代码保存到 N-A 防御服务器中磁盘的对应路径下。图 9 所示为实验过程中 crash 目录下生成的保存能够导致服务器程序崩溃的恶意代码内容。

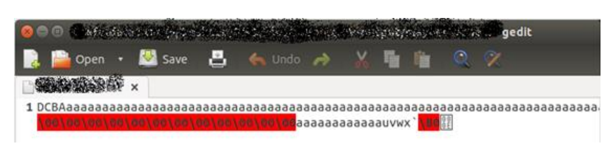



图 9　造成系统崩溃的数据

同时，防御系统还会向系统管理员发送简易报告。图 10 所示是实验中产生的一条针对 Tftp32 的恶意代码的报告示意图。

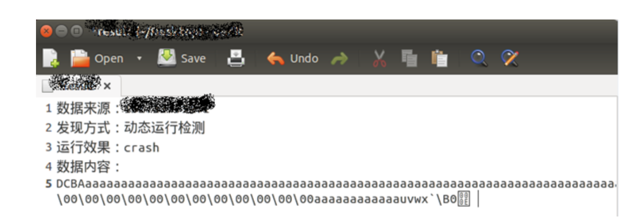



图 10　防御报告

图 11 所示为“防火墙＋N-A 防御服务器”策略下，针对两种网络程序的恶意代码攻击防御效果。可以看出，单纯防火墙策略的防御效果较弱，特别是在恶意代码的特征码未更新到防火墙策略库时，针对两款软件漏洞的防御率仅能达到 97.78%和 98.0%。

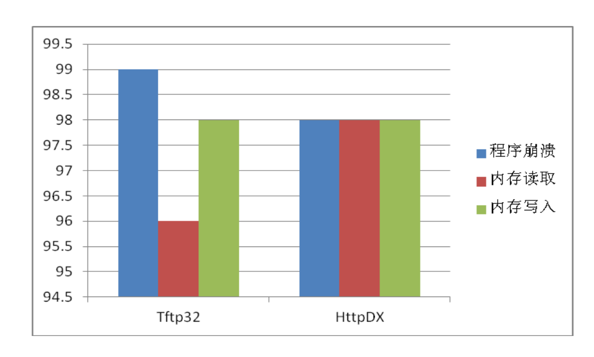



图 11　实验防御效果统计

实验证明，如果不通过 N-A 防御服务器的检测，单纯采用防火墙检测，则检测成功率较低，防御效果较差,而采用“防火墙+检测系统”的检测策略，则分别实现了 97.78%（HttpDX）和 98.0%（Tftp32）的恶意代码的拦截成功率。

## 5　结束语

本文提出了一种动静结合的网络数据检测技术，并在该技术思想的基础上实现了一套网络数据检测系统。通过具体实验证明，该系统在运行过程中通过静态分析检测和动态运行检测相结合的方式，提高了恶意代码检测的成功率，特别是提高了对一些利用未知漏洞的恶意代码实现防御的成功率。该系统在静态检测过程第二阶段的数据检测算法还有改进空间，在检测过程中需要大量的存储空间记录信息存在一定冗余。下一步将针对该问题寻找更加可靠和空间占用小的算法。

## 参考文献：

[1] Wang Shiwei. On information security, network security and cyberspace security [J]. Journal of Library Science in China, 2015 (2): 72-84.

[2] Slater D. Social relationships and identity online and offline [M]// Lievrouw L, Livingstone S. Handbook of New Media: Social Shaping and Consequences of ICTs. London, UK: Sage Publications, 2002: 533-546.

[3] DaveAitel. The advantages of block-based protocol analysis for security testing [R]. [S. l. ] : Immunity, Inc. 2003.

[4] Hundley R O, Anderson R H. Emerging challenge security and safety in cyberspace [EB/OL]. [2016-07-05]. https://www. rand. org/content/dam/rand/pubs/monograph_reports/MR880/MR880. ch10. pdf.

[5] 魏为民，袁仲雄．网络攻击与防御技术的研究与实践 [J]．信息网络安全, 2012 (12): 53-56.

[6] 汤龙．计算机网络防御策略求精关键技术 [J]．信息系统工程, 2015 (10): 65-65.

[7] Richard D K, Richardson R T, Ludgate L D, *et al.* Computer network

monitoring with test data analysis: US, US 6931357 B2 [P]. 2005.

[8] Li L, Hu Z Y. The research of data stream technology in computer network security monitoring [C]// Proc of International Conference on Intelligent Systems Research and Mechatronics Engineering. 2015.

[9] Corchado E, Álvaro H. Neural visualization of network traffic data for intrusion detection [J]. Applied Soft Computing, 2011, 11 (2): 2042-2056.

[10] Ingham K, Consulting K I, Forrest S. A history and survey of network firewalls [C]// Proc of the 5th Usenix Unix Security Symposium Usenix Association. 2002: 1-42.

[11] 彭沙沙，张红梅，卞东亮. 计算机网络安全分析研究 [J]. 现代电子技术, 2012, 35 (4): 109-112+116.

[12] Salehfar H, Zhao R. A neural network preestimation filter for bad-data detection and identification in power system state estimation [J]. Electric Power Systems Research, 1995, 34 (2): 127-134.

[13] 任午令，赵翠文，姜国新，等. 基于攻击行为预测的网络防御策略 [J]. 浙江大学学报：工学版, 2014, 48 (12): 2144-2151.

[14] Watson P T, Swix S R, Gray J H. Set top box with firewall: US, US20040237098 [P]. 2004.

[15] Orebaugh A, Ramirez G, Burke J, *et al.* Wireshark & ethereal network protocol analyzer toolkit [M]// Jay Beales Open Source Security. 2014: 523–540.

[16] Thottan M, Ji C. Adaptive thresholding for proactive network problem detection [C]// Proc of the 3rd IEEE International Workshop on Systems Management. 1998: 108-116.

[17] Carpenter B E, Partridge C. Internet requests for comments (RFCs) as scholarly publications [J]. ACM Sigcomm Computer Communication Review, 2010, 40 (1): 31-33.

[18] Button K. FireEye Acquires iSight in Cybersecurity M&A [J]. Mergers & Acquisitions the Dealermakers Journal, 2016.